

SMASHUP: Secure Mashup for Defense Transformation and Net-Centric Systems

Mark D. Heileman^{*†a}, Gregory L. Heileman^{†b}, Matthew P. Shaver^c,
Mike Gilger^{†a}, Pramod A. Jamkhedkar^b

^aModus Operandi, Inc., 709 S. Harbor City Blvd., Suite 400, Melbourne, FL 32901;

^bUniversity of New Mexico, Dept. of Electrical & Computer Engineering, Albuquerque, NM 87131;

^cAir Force Research Laboratory, AFRL/RIEBB, Rome, NY 13441

ABSTRACT

The recent development of mashup technologies now enables users to easily collect, integrate, and display data from a vast array of different information sources available on the Internet. The ability to harness and leverage information in this manner provides a powerful means for discovering links between information, and greatly enhances decision-making capabilities. The availability of such services in DoD environments will provide tremendous advantages to the decision-makers engaged in analysis of critical situations, rapid-response, and long-term planning scenarios. However in the absence of mechanisms for managing the usage of resources, any mashup service in a DoD environment also opens up significant security vulnerabilities to insider threat and accidental leakage of confidential information, not to mention other security threats. In this paper we describe the development of a framework that will allow integration via mashups of content from various data sources in a secure manner. The framework is based on mathematical logic where addressable resources have formal usage terms applied to them, and these terms are used to specify and enforce usage policies over the resources. An advantage of this approach is it provides a formal means for securely managing the usage of resources that might exist within multilevel security environments.

Keywords: mashup, usage management, access control, multilevel security, wiki, Web 2.0.

1. INTRODUCTION

The ability to express fine-grained usage policies over the resources that exist within multilevel security (MLS) environments, coupled with the ability to securely enforce these policies, would greatly enhance the capabilities of Department of Defense (DoD) analysts working within these environments. In particular, the availability of such services would allow these analysts to more easily combine and share appropriate information with others that should be allowed to view or use this information, as determined by their security credentials and “need to know.” In order to accomplish this goal, this paper introduces the notion of integrating a fine-grained usage management framework into MLS environments. The granularity of the usage policies supported by this approach is at the level of any identifiable/addressable resource or service (both will be referred to as simply a “resource”) that might exist within the MLS environment, and the policies themselves may take into account any aspect of who is using a resource, i.e., their

* mheileman@modusoperandi.com; phone 1 (321) 473-1420; fax 1 (321) 473-1499; www.modusoperandi.com.

† Supported in part by the AFRL under Small Business Innovation Research Contract No. FA8750-10-C-0090. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the AFRL. Publication clearance number: 88ABW-2011-1781.

© Copyright 2011 Society of Defense, Security & Sensing. One print or electronic copy may be made for personal use only. Systematic reproduction and distribution, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper are prohibited.

Citation: Mark D. Heileman, Gregory L. Heileman, Matthew P. Shaver, Mike Gilger, and Pramod A. Jamkhedkar, "SMASHUP: secure mashup for defense transformation and net-centric systems", Proc. SPIE 8062, 806206 (2011); doi:10.1117/12.883486. Permalink: <http://dx.doi.org/10.1117/12.883486>.

credentials, as well as the specific environments in which a resource may be used, along with how it might be combined or used with other resources.

In order to demonstrate the feasibility of this approach, a prototype demonstration system was constructed that integrates a previously developed usage management framework into a type of networking infrastructure that exists in MLS environments. The resulting system is termed SMAPSUP, as it provides the ability to perform secure mashups in MLS environments. This is demonstrated by deploying SMASHUP within a Semantic MediaWiki application. This paper describes the research associated with the development of the SMASHUP prototype. The remainder of this paper is organized as follows. This section introduces the subject by way of its background, its significance, and a review of pertinent literature. Section 2 provides a description of the methods, assumptions, and procedures used in our investigation. Section 3 provides a detailed description of the demonstration system that was built as a result of this research. Finally, Section 4 concludes with an interpretation of our findings and our intent for future research.

1.1 Background and Motivation

Existing and new DoD systems are exposing more and more information via web service techniques and technologies. However usage of these services has not extended much beyond the developer community as the technical skills and tools needed to invoke these services have remained out of reach of the average end user. Additionally the certification and accreditation of such tools discourages and/or prevents the average end user from modifying the manner in which information reaches them due to concerns regarding confidentiality, integrity, and availability.

The recent development of Web 2.0 technologies now enables users to easily mashup data from a vast array of different information sources available on the Internet. The ability to harness and leverage information in this manner provides a powerful means for discovering links between information, and greatly enhances decision-making capabilities. The availability of such services in a DoD environment will provide tremendous advantages to the decision-makers engaged in analysis of critical situations, rapid-response, and long-term planning scenarios. However in the absence of mechanisms for managing the manner in which mashups are performed and subsequently used, any mashup service in a DoD environment also opens up significant vulnerabilities to insider threat and accidental leakage of confidential information, just to name a few security issues.

1.2 Overview

In this research project, we have developed a framework that allows integration via mashups of content from various data sources in a secure manner. The framework makes use of a license management mechanism that was developed as part of this research. A license can be used to wrap usage policies around any addressable resource (i.e., data or services), thereby specifying rules over the manner in which the resource may be used in different environments. We capture the environment itself in the form of context, and supply this context to a usage management mechanism that considers a license within a given context prior to making any usage decisions. The usage management mechanism can then be equipped with different frameworks for reasoning over the usage terms contained in the license and context, including ones based on formal logic theory. Because of this formalism, we are able to analyze a given usage management setting, and determine what can and cannot be decided within that framework. It is important to recognize that different content “ecosystems” have different needs in terms of usage management. For example, one would expect that the usage management requirements associated with e-books to be quite different from those associated with documents in a multilevel security environment. Thus, we have created a system that allows these domain-specific usage terms to be separated out into what we call the *Ecosystem Ontology*. This effectively separates the mechanisms that have been developed for managing licenses and reasoning over usage scenarios from the particular ecosystem in which they are being used. This makes it very easy to add new usage management functionality to the existing framework.

1.3 Literature Review

Recently, there have been a number of papers that consider security issues related to mashups. For instance, a number of groups have investigated the problem of cross-domain scripting in mashups [1; 2]. This problem arises from the browser security model that is used in typical mashups, allowing the scripts associated with a mashup source to change the data that was supplied by other mashup sources. Another architecture was proposed that considered how to support authentication and authorization when dealing with non-public data in mashups [3]. In this case, an application programming interface (API) was provided that authenticated a user via OpenID, prior to providing data to that user. We are not aware of any research that deals with the application of usage management to mashups.

Before considering how to incorporate usage management into mashup applications, it is instructive to consider the types of mashups that exist. In a *client presentation mashup*, the mashup program receives formatted HyperText Markup Language from data sources, often called widgets, and plugs them into the mashup program's web pages. A mashup editor is a tool that is typically used to create these mashup applications. Such editors are now part of the Google App Engine product [4], Yahoo! Pipes [5] and IBM Mashup Center [6], just to name a few. Figure 1 shows the type of graphical programming environment that is typically provided by these mashup editors. This mashup was designed to use eBay's Really Simple Syndication API in order to find specific items within a given price range. In the example, a URL Builder module is used to create a query to eBay. A text input is "wired" to this module in order to provide the search term, and then a number of additional modules are connected in order to filter the search results according to the user's criteria. A mashup editor is part of the Google App Engine product [4]. In *client service mashups*, the mashup program also receives data from different sources; however, these sources are processed within the browser using JavaScript prior to presenting the mashed-up information to the end user. Finally, in *external service mashups*, a server collects data from different sources, and then processes it using a scripting language such as Ruby, PHP, or Python, prior to presenting the mashed-up information to the end user.

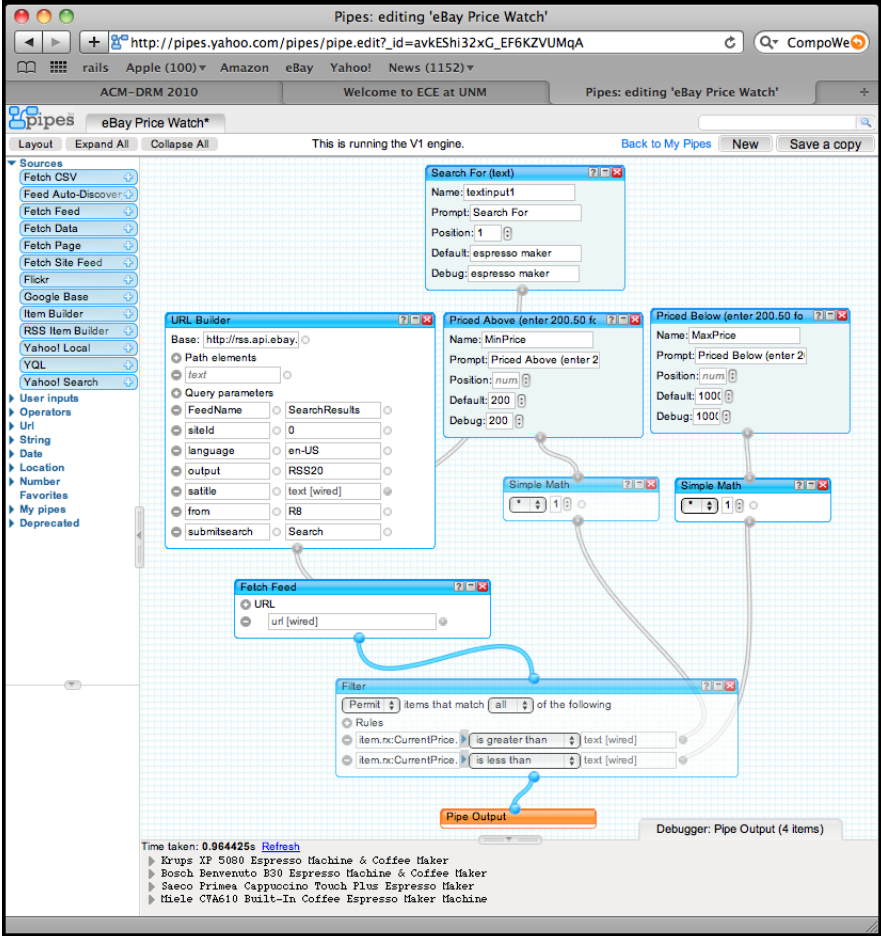


Figure 1: The user interface provided by Yahoo! Pipes.

Usage management is the management of the usage of resources (and data) across and within computing environments. Traditionally, usage policy specification has been the responsibility of the owner of the computing environment. The use of resources (and data) has generally been restricted to closed computing environments that are managed by a single entity. However, with the explosion of Web 2.0 applications on the Internet, these assumptions no longer hold. The separation of resource owner and the owner of the computing environments within which the resources are consumed is more pronounced. Moreover, computing environments within which resources are used are no longer fixed, and resources often move across multiple computing environments.

Most of the current research in areas related to usage management has been directed towards developing a more expressive language by using either a different type of mathematical logic, or a formalism with a greater reasoning capability [7; 8; 9; 10; 11; 12; 13]. Such advancements, even though useful in closed systems, fail to address the interoperability challenge posed by open environments. The solutions that have tried to address interoperability have often resorted to translation mechanisms, where the entire policy is translated to a different language [14; 15; 16]. Such translations are infeasible, and difficult to carry out for most of the policy languages [17; 18]. Other approaches have led to the development of complex policy specification languages that have tried to establish themselves as the universal standard [19; 20; 21; 22]. Such an approach requires standardization of the complete policy language, which stifles innovation and flexibility [14; 23; 24; 25].

2. METHODS, ASSUMPTIONS, AND PROCEDURES

In a typical access control setting in a DoD computing environment, the usage of information is tightly constrained through policies that are also tightly coupled to the computing environment through centrally managed policies. On the public Internet, however, information moves across highly networked, distributed computing environments that are not a part of a single centrally managed system. This enables the information passed around on the Internet to be used in increasingly innovative ways through the transformation, processing, and merging of this information with other information across disparate computing environments. This capability enables the mashup process, where information from different sources is merged in order to generate a new information source. This process, however, necessitates usage management policies to be tightly coupled to the resource, rather than the system as in the case of most DoD environments. In this research, we enable usage policies to be interpreted and enforced as the resource moves across different computing environments, thereby enabling the mashup capability in multilevel security environments.

With traditional security systems, the responsibility of usage policy specification has been with the owner of the computing environment, and the use of resources (and data) has generally been restricted to closed computing environments that are managed by a single central entity. However, in order to take advantage of Web 2.0 capabilities in security environments, these assumptions must change. The separation of the resource owner from the computing environment must be supported, so that these resources can more easily move securely across multiple computing environments. Below we describe a usage management framework that allows for this capability. Following a description of the framework, we define a calculus that provides a platform upon which different usage control specification languages can be developed without worrying about system-specific dependencies. These languages can be used to generate smart licenses that can interoperate with different computing environments. One of the key features of the calculus is that it formalizes the notion of interoperability and therefore allows for formal reasoning about interoperability. Such a feature is essential for automated use of resources across different security networks. The calculus also enables dynamic interpretation of policies, which allows a given policy to be interpreted in different ways within the same computing environment, depending upon circumstances that may be changing within that environment.

2.1 A Usage Management Framework

In its simplest form, usage management involves usage policies, expressed in the form of a license, which are interpreted and enforced within a computational environment.

In order to simplify the understanding of the framework, we divide usage management into two stages, namely, the setup stage and the working stage. The setup stage includes the process of setting up the computational environment and the generation of a license. The working stage includes the process of license interpretation and enforcement within the computational environment.

Usage Management – Setup Stage

The setup stage involves the generation of a license object and a context object. Both license and context objects have a behavior, maintain an internal state and a standard interface. The setup phase for usage management is shown in Figure 2. The figure shows two entities, a license and a context, being created by a license generator and usage management mechanism, respectively. A license generator is a program that allows a resource owner to express licenses, and can be thought of as a service (e.g., provided as a web service). The usage management mechanism is a program that manages usage on the client machine where the resource is being used. The license generator and usage management mechanism use a common ecosystem ontology in order to create a context and license.

Ecosystem Ontology and Interoperability. In order for a license to be interpreted within a computational environment, there needs to be an agreement on the terms expressed in the license and the ones defined within the computational environment. That is, a common ontology needs to be used for the creation of both a license and a context. Such an ontology defines a common vocabulary for agents or programs that need to share information in a domain, and includes machine-interpretable definitions of basic concepts in the domain, as well as the relationships among them.

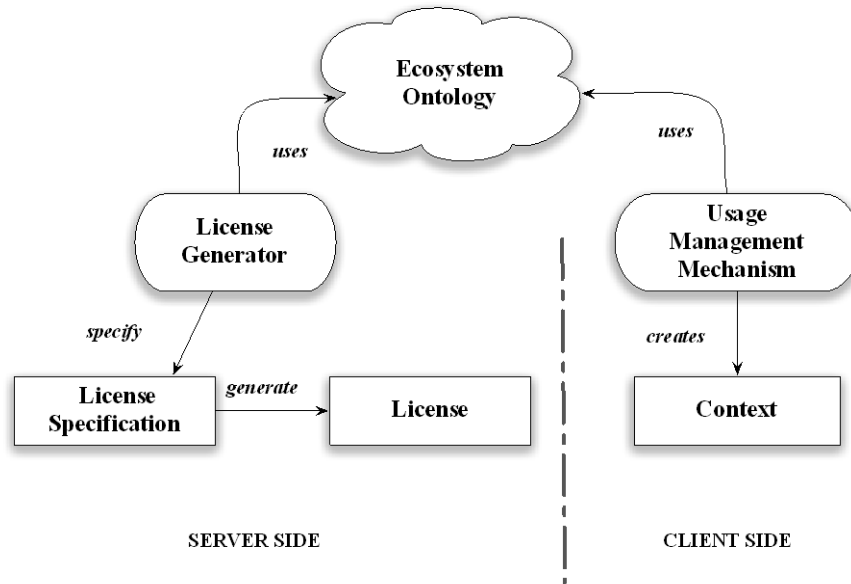


Figure 2: The setup stage for usage management.

Context. As mentioned previously, a context is a formal representation of the computational environment in which a resource will be used. More formally, a context object is an instantiation of a context that captures the structure and state of a computing environment. Throughout this paper, we refer to a context object as simply the context. A context is structured according to the ecosystem ontology, and it represents the entities within a computational environment, as well as the relationships among these entities. For a given computational environment, each entity is defined by a set of attributes, and the context maintains the current values for each of these attributes. The primary goal of the context is to capture the conditions under which actions are carried out within a computational environment. The context also maintains an interface that supports the following functions:

1. Update and retrieve relationships among the entities.
2. Update, retrieve, and compare operations over the attribute values for a given entity.

Depending on the type and complexity of a given context, it may support different types of interfaces.

Smart Licenses. The manner in which we express and interpret licenses within the framework is significantly different from the way licenses are used. For example, in traditional rights management systems, policies are expressed within a license by means of a policy specification language. The language can either be a descriptive natural language (e.g., creative commons), a machine readable Extensible Markup Language (XML)-based language (e.g., eXtensible Rights Markup Language (XrML) or Open Digital Rights Language (ODRL)), or a logic-based language with a formal syntax and semantics. In all of these cases, the license is expressed in terms of a descriptive statement, and an interpreter that understands the syntax and semantics of the descriptive license must exist within the computational environment. Thus, with the traditional approach, a license is descriptive and passive, and it is the responsibility of the interpreter to interpret the license within the computational environment.

In our approach, as shown in Figure 2, the descriptive, passive license is transformed into an executable, active license object, which we call a smart license. More specifically, a smart license is an executable object that has a behavior, a state, and an interface.

The behavior of a license reflects the expressive complexity of the policy the license represents. Different licenses can capture different policy language complexities. Furthermore, different types of license objects can capture various usage semantics, such as, permissions, obligations, temporal dependencies, partial dependencies, and interleaving semantics. The operation of the framework is agnostic to the behavior of a license object, which in our approach is hidden inside the license object itself. This enables license descriptions from different policy languages to be transformed into license objects, and interoperate within the framework we have created.

The state of a license object captures the history of resource usage associated with the license. Every time usage of a resource associated with a license is performed within a computational environment, the license object updates its state in order to record the history. Maintaining the usage history in the form of a state within the license object is an important design decision that allows managing licenses that are used across multiple computing environments, and allows for a decentralized approach to usage management.

The interface to a license object allows the usage management mechanism to query the license object regarding usage. Different types of license will support different levels of interfaces. The “smarter” a license type, the richer the interface it will support. For example, the interface provided by a license object might allow for:

1. Querying a license regarding decisions on usage.
2. Enabling update, retrieve, or reset of the license state.
3. Checking compatibility with another license or merging with another license.

Next we describe the working stage of the usage management framework – that is, the interactions among the various usage management entities that operate on the client side, where the resource is being used.

Usage Management – Working Stage

The working stage of the usage management framework involves interpretation and enforcement of a license within a computing environment. Figure 3 shows the working of the usage management framework. The components that are shown in grey are considered parts of the usage management framework. The components that are shown in white are supporting components that are external to the usage management framework. That is, the usage management framework uses the services provided by the white components.

The working stage of usage management framework consists of four components, namely, the license, context, action map, and usage management mechanism. The usage management entities operate within the computing environment where the license needs to be interpreted and enforced. The behavior and interface of a license and context have been previously explained. We now consider the usage management components in this figure in more detail.

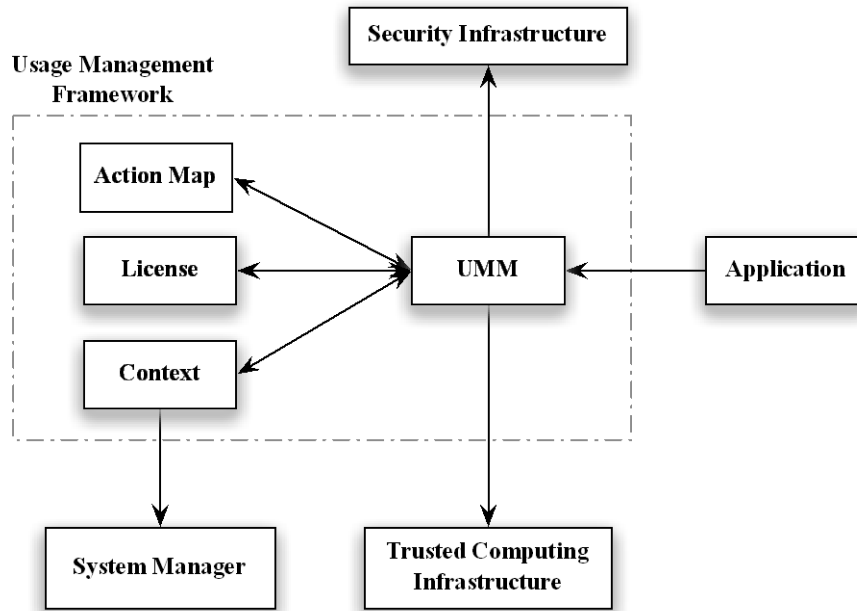


Figure 3: The working stage for usage management, where UMM denotes the usage management mechanism.

Action Map. Different types of usages can be expressed using verbs such as “play”, “view”, “print”, and “loan”, etc. When a license is created in order to define a usage policy, it must use this vocabulary to express the policy terms. Each of these verbs has a specific interpretation depending upon the computing environment within which it is to be interpreted.

In every computing environment, actions are carried out as discrete identifiable events. These events are different for each computing environment, and may depend on the type of computing platform, operating system, etc. Hence, every verb can be interpreted within a computing environment in terms of a single event or a sequence of such events. We use the term “activity” for the verbs used in a license, and the term “action” for the identifiable events in the computing environment.

The separation of verbs used in licenses and their interpretation within computing environments allows for dynamic interpretation of licenses, and will be discussed in more detail later in the paper.

Usage Management Mechanism. This component acts as a controller that manages communication between the components of usage management framework, as well as communication with external services. The usage management mechanism also provides an interface to the user-level application, allowing it to manage usage. Different applications can use the common interface provided by the usage management mechanism, and the usage management mechanism is also capable of interacting with different types of licenses via their standard interfaces.

System Manager. This component is responsible for providing the values of current system attributes. More specifically, a system manager is a computation-environment-specific piece of software that provides a service to the context object by making available the current values of the system.

Trusted Computing Infrastructure. This component is a service that may be provided by the computing environment enabling enforcement of usage decisions provided by the usage management mechanism. That is, the trusted computing infrastructure is a system specific service that may be able to enforce whether or not the execution of actions in the computing environment may take place.

Security Infrastructure. This component includes services that allow for license management. Such services allow the usage management mechanism to manage authentication of users, validation of licenses, encryption and decryption of licenses, and other such license management tasks. The security infrastructure is generally computation-environment-specific and provided as a service to the usage management mechanism.

In the following workflow we define the steps involved in a typical use case of policy interpretation and enforcement, and consider how information flows between the different components shown in Figure 3. This workflow is different from the operational mode, which defines the manner in which policy decisions are carried out. In this workflow sequence we assume that context values are checked by the policy before the activity is carried out, and that state is updated after the activity is carried out:

1. The application queries the usage management mechanism in order to determine whether or not an action (or set of actions) is permitted. The application provides subject attributes and resource attributes.
2. The usage management mechanism and system manager update the context object with subject attributes, resource attributes, and the computing environment attributes.
3. The usage management mechanism queries the action map for the activity (or a set of activities) that corresponds to the action supplied by the application.
4. The usage management mechanism queries the license in order to determine whether or not the activity is permitted.
5. The license queries the context in order to determine the conditions under which the activity can be performed. The conditions may include subject attributes, resource attributes, and computing environment attributes.
6. Depending upon the usage history, and the present conditions under which the activity is to be performed, the license object determines whether or not an activity can be performed, and informs the usage management mechanism.
7. The usage management mechanism informs the application about the decision and uses the services of the trusted computing infrastructure to enforce the decision.
8. If the action is indeed carried out, the license object is notified about the execution of the action.
9. The license object records the event by updating its state.

We now consider a formal model for the usage management framework we have just described.

2.2 Formal Model for Usage Management

A formal model for usage management was developed as part of this research effort and a high level description of it is provided next. The primary entities in the usage management model are the context, the license, and the usage management mechanism. The *context* captures the reference within which usage management is being carried out. E.g., a context models the computing/system environment where resource usage is occurring, the agents operating in the system, and the resources within the system. A *license* consists of the usage rules that specify the manner in which the agents may use the resources in the system. The usage policies associated with a particular resource within a given DoD computing infrastructure would be encoded in a license. Finally, the *usage management mechanism* interprets and enforces a license within a given context. Each of these entities is described more formally in detail below.

A context is defined by the tuple $C = \langle E, S, R \rangle$, where E , S , and R represent the set of system, environment, and subject properties, respectively. Usage policies are defined over these properties and stored in a license. A license is defined by the three-tuple $lic = \langle ra, e, i \rangle$, where ra defines the set of restricted activities in the license, e is a set of license restrictions defined over ra , and i is the set of interface functions supported by the license. An activity is a license abstraction for the different operations that may be carried out, e.g., view, print, etc., and a restricted activity sets the context under which these activities are allowed. A license expression provides usage semantics, e.g., permissions, obligations, rights, etc., to restricted activities, and is assumed to use a particular type of logic (e.g., first order predicate logic). Finally the license interface provides a set of functions that the usage management mechanism uses to query the license in order to determine if particular uses should be allowed. This involves interpreting whether or not a particular requested action can be satisfied by the available restricted activities. The usage management mechanism operates within a particular computing environment, and is also responsible for enforcing usage according to what uses are allowed or not allowed.

We have defined an operational calculus over the model described above. The semantics of this calculus are depicted at a high level in Figure 4 by showing the generic steps that take place in order to enforce usage terms within a particular context.

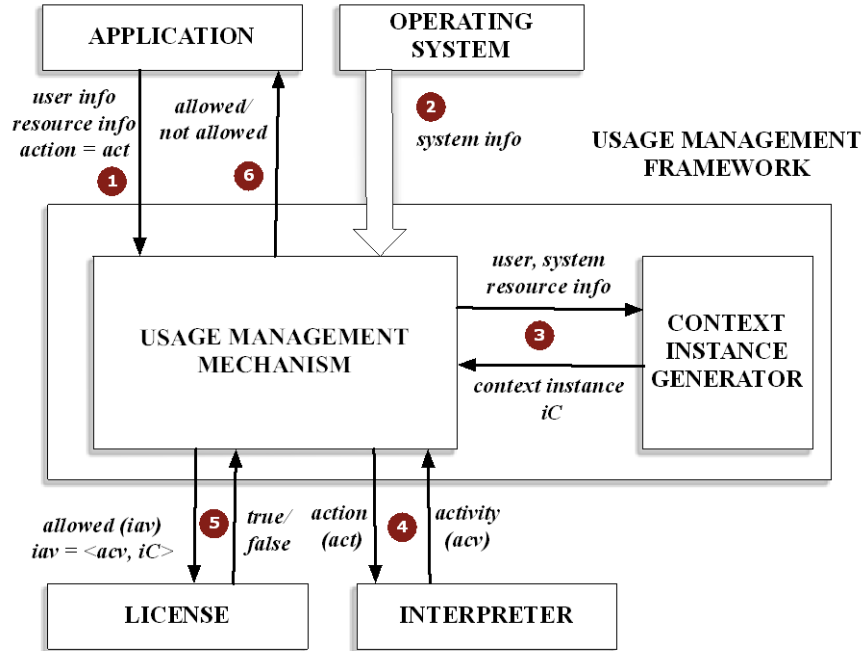


Figure 4: The operational semantics of the calculus associated with the formal model for usage management.

Specifically the process of license enforcement takes place in six steps. In step 1 the application queries the usage management mechanism about whether or not a particular action (*act*) on a given resource by a given user is valid. As a part of the request, the application provides information about the user, the resource, and the action that is to be performed. In step 2 the usage management mechanism obtains the current state of the computing environment from the operating system. The type of information obtained by the usage management mechanism depends upon the manner in which the context is modeled. This information may include current location, day, date, time, IP address, etc. In step 3, the usage management mechanism generates the context instance *iC* by using the current values of system parameters, user information, and resource information. This is followed by step 4, where the usage management mechanism queries the interpreter to determine the activity (*acv*) that corresponds to the act information provided by the application. Once the activity corresponding to the action is obtained, the usage management mechanism generates the activity instance *iav = <acv, iC>*. In step 5 the usage management mechanism invokes the *allowed(iav)* function provided by the license, the license executes an *allowed(iav)* function, and returns a Boolean value to the usage management mechanism. Finally, in step 6, the usage management mechanism conveys to the application the validity of the action.

The model for reasoning over usage decision is currently based on simple first order logic, and does not define any particular policy specification language. Rather, the model provides a scaffolding upon which different types of usage management policy languages can be developed, and still operate within the usage management framework. The design of the model allows for a formalization of the notion of interoperability of licenses across computing environments, and essential requirement for secure mashups.

Since the model supports formal definitions for license interoperability, it is possible to manage automated distribution of licenses across different computing environments. In order to achieve this, a standard interoperability protocol must exist between licenses and usage management mechanisms as shown in Figure 5.

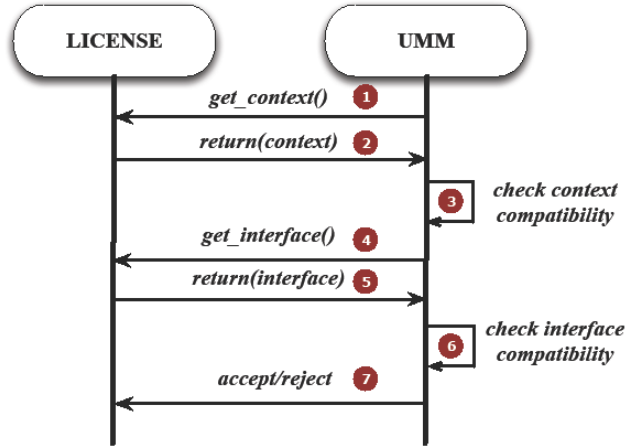


Figure 5: The protocol defined between a license and the usage management mechanism (UMM).

The protocol works as follows:

In messages 1 and 2, the usage management mechanism obtains the context that is used by the license to express the policy. In message 3, the usage management mechanism checks context compatibility. In messages 4 and 5, the usage management mechanism obtains the set of license interface functions, and checks interface compatibility in message 6. If both the context and interface compatibility checks pass, then the usage management mechanism sends an accept message, else it sends a reject message to the license.

Automated reasoning of interoperability is another distinguishing feature of the model. It should be noted that interoperability is a not an add-on capability in our approach, but a feature that is supported by the calculus from within.

3. RESULTS AND DISCUSSION

The SMASHUP demonstration scenario is shown in Figure 6. It involves cross-domain interactions through a guard. A guard is a specialized software application that runs in hardened computer or trusted computing environment between two security domains, and allows only data that meets certain criteria to pass from one domain to another. Fourteen data sources (including map data, route data, blue force data, open source data, imagery data, weather data, and logistics data) were created for the demonstration. Appropriate licenses for each of these data sources were created, and linked to these data sources. Finally, the context of three particular users (namely Coalition Partner, Intel Analyst, and Supply Officer) was set in the demonstration according to their role, security clearance, projects, domain, operating system, and network device. The License Manager used this “mashup” context in order to determine the particular data sources that could be provided to particular users. The manner in which this is accomplished is discussed in Section 3.1 below. These elements were all integrated in the demonstration.

Key elements of the SMASHUP demonstration usage management which are shown in Figure 6 are:

(1) Policy & DA Registry. Policies (encoded in licenses) as well as Data Assets (DAs) are stored in the Registry. The policies define access and usage control rules in contexts such as “data” and “mission.” DAs define their capabilities so that the Policy Manager can determine if a specific DA has the capability of assuring specific Policy rules can be enforced.

(2) Policy Manager. The Policy Manager interprets the policies as well as the capabilities of the various DAs to enforce the policies as applied to a specific user context. Note that the Registry and the Policy Manager can be implemented as a single unit.

(3) GUI Component. Used as an interface between the Policy Manager and the graphical user interface (GUI) for creating mashup content. Provides dynamic GUI hints to prevent the user from trying to create a mashup that fails a security policy. For instance, if two specific DAs can’t be used at the same time, if the user selects one of those DAs, the GUI Component will automatically disable the other one.

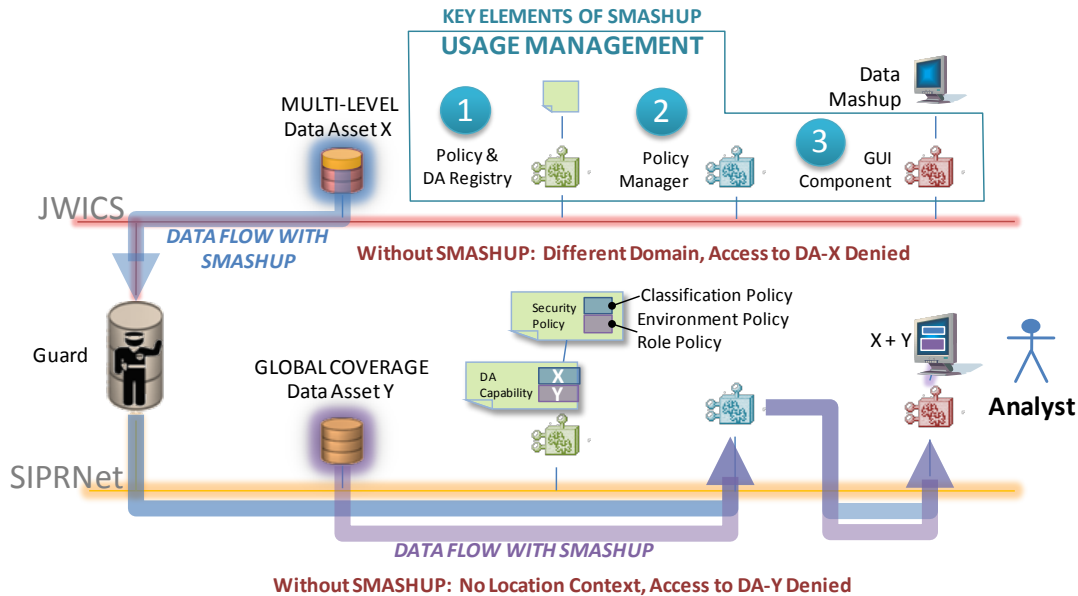


Figure 6: The demonstration scenario involving cross-domain interactions through a guard.

In the use case of Figure 6, a DA with multilevel data is on a different security domain than the user (i.e., analyst) desiring to mashup data. Without a way to enforce policies across the domains, typically, that DA will not be accessible to the mashup user – even if the DA contains data that is both appropriate and at the right security level for the user. With SMASHUP, the policies flow across domains, so if both policies align with the domain environment security policy requirements, then the data from the DA will move across the Data Guard to the domain where the mashup user is.

3.1 License Server

The precise steps that were implemented in the deployed license server are shown in Figure 7. First, as shown in the figure, a resource must register with the license server. This involves providing the license server with a set of license terms that deal with how that resource may be used.

The license server then creates a license, stores it locally in its license store, and returns an identification (ID) to the resource that can be used to identify the license at a later time. In Figure 7 we show two such resources, and it is important to recognize that these may be any addressable resource, including addressable services. Next, if a mashup program would like to use these resources, it must contact them in order to obtain their license IDs, as shown in Figure 7. Next, the mashup program contacts the license server, possibly providing its own set of license terms, along with the source IDs. The license server makes use of the license evaluator in order to create a license that satisfies all of the terms associated with the sources in the mashup, and the mashup program itself stores the license in the license store, and returns a license ID to the mashup program. This is shown in the bottom portion of Figure 7. It should be noted that, given the manner in which the license server was designed, the output of the mashup program may now be provided as a source, and the usage terms associated with sources that may be buried many layers deep within such a nesting will continue to be asserted. In addition, the reasoning that is currently provided for creating a combined license from the licenses of the sources and the mashup program is trivial. Currently, we simply provide a union over all the license terms associated with these entities.

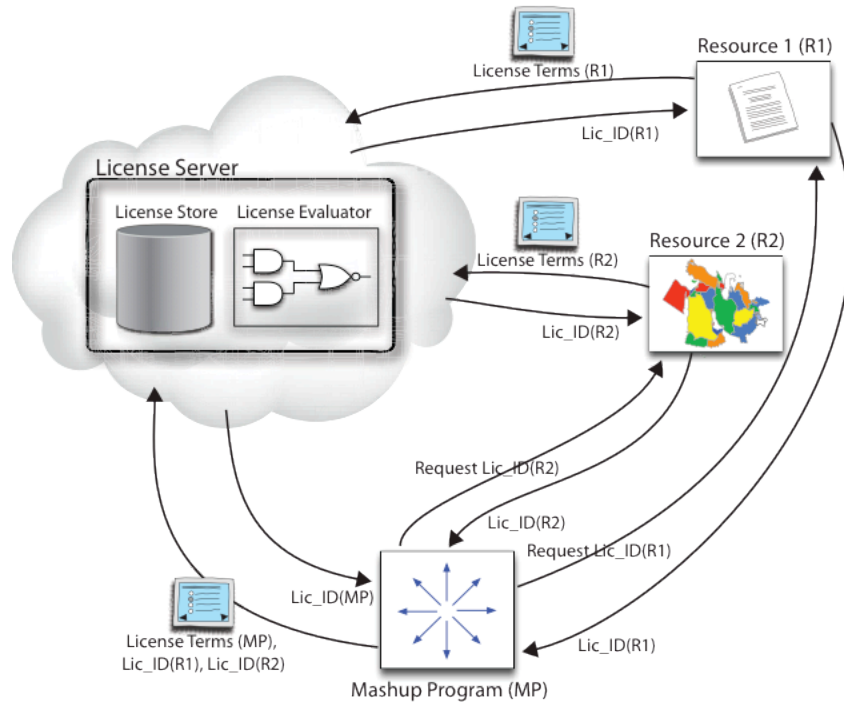


Figure 7: The steps necessary to set up the license server.

The protocol for using the mashup program is shown in Figure 8. A browser navigates to the mashup program, and requests the services provided by the mashup program. The mashup program may collect information about the browser, user, etc., and then forwards this context to the license server. The context itself is encoded as part of the web session. Next, the mashup program requests data from the sources, providing its license ID to each source. The sources provide this license ID to the license server, and based upon the current context of the mashup program, the license server employs the license evaluator in order to provide an authorization to each source. Depending upon what the license server authorizes, different data will be mashed-up and provided to the end user's browser.

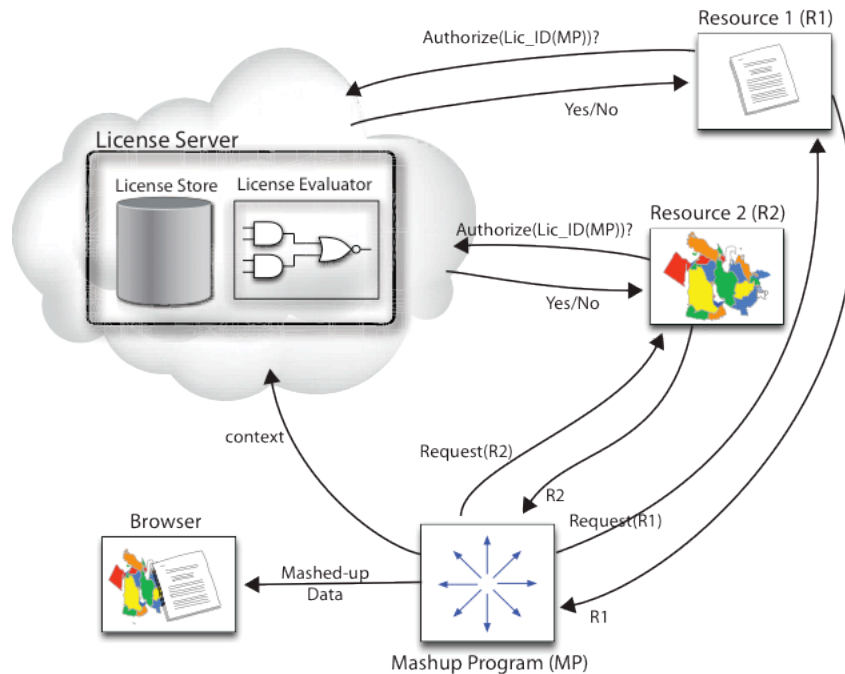


Figure 8: The protocol for using the license server in a mashup.

3.2 License Management Application

The license management application has three main functionalities associated with it: resource management, license management, and the mashup mechanism.

The resource management functionality shows the resources that have been registered with the system, along with their physical storage location. In addition, certain properties associated with the resource are shown, e.g., the security classification for the resource, and whether or not it contains map information, or information that could appear on a map. A RESTful (a simple web service implemented using HTTP and the principles of REST) interface is provided for managing these resources.

The license management functionality provides a listing of the currently defined licenses. This interface is used to perform license binding. In particular, by selecting the “New license upload” option, it is possible to bind a license (stored at a particular location) to a specific resource. For this application the licenses are specified using XML, but other formats are possible. Currently, these licenses must be constructed by hand; however, in the future, we will develop an application that allows licenses to be easily constructed for the multilevel security usage domain.

Finally, the mashup mechanism interface allows one to specify properties about a user and their current environment (i.e., their context), and upon executing the “Get Resources” function, the IDs of the resources that this user is able to view within their current context is returned. Note that the usage management mechanism, referenced in Figure 4, is consulted in order to determine what resources IDs should be returned.

In the demonstration application described below, these resource IDs are returned programmatically (accessible via a RESTful interface); however, this license management application is useful for debugging purposes, as it allows one to quickly view the resource identifications that are applicable to particular mashup scenarios.

3.3 Demonstration Application

We created an initial graphical user interface for the limited prototype demonstration. The GUI was constructed with the Semantic MediaWiki application. MediaWiki is a web-based wiki software application developed by and used on all projects of the Wikimedia Foundation, as well as on many other wiki websites worldwide. A wiki (a type of Web 2.0 application) is a website that allows the easy creation and editing of any number of interlinked web pages via a web browser using a simplified markup language or a WYSIWYG text editor. Semantic MediaWiki is an open-source extension to MediaWiki that lets one store and query data within the wiki's pages [26].

4. CONCLUSIONS

In this research we developed a formal model for a flexible usage management framework that operates in open distributed environments. The underlying principles of the proposed model include the clear separation of policy expression, policy interpretation, and policy enforcement. This approach allows policies to be expressed with minimal a priori knowledge of the computing environments in which the policies will be interpreted. The model supports many important features such as dynamic interpretation, and allows us to formalize the notion of license interoperability. These features are critical for expression, interpretation, and enforcement of licenses in distributed environments, where multiple computing systems across a network must be able to interpret the license.

As future research we intend to investigate the use of different types of logics with varying capabilities within the usage management model. For the purposes of this initial effort, we have included a very simple logic, but this can be easily extended to create much more powerful reasoning capabilities within the current usage management model. Finally, we intend to define in greater detail the content ecosystem associated with DoD multilevel security environments.

REFERENCES

- [1] *CompoWeb: A Component-Oriented Web Architecture*. **Guo, Rui, et al.** Beijing : Association for Computing Machinery, 2008. Proceeding of the 17th international conference on World Wide Web. pp. 545 - 554. 978-1-60558-085-2.
- [2] *Subspace: secure cross-domain communication for web mashups*. **Jackson, Collin and Wang, Helen J.** Banff : Association for Computing Machinery, 2007. Proceedings of the 16th international conference on World Wide Web. pp. 611 - 620. 978-1-59593-654-7.
- [3] *A design of usable and secure access-control APIs for mashup applications*. **Hashimoto, Ryota, Ueno, Nachi and Shimomura, Michio.** Chicago : Association for Computing Machinery, 2009. Proceedings of the 5th ACM workshop on Digital identity management. pp. 31 - 34. 978-1-60558-786-8.
- [4] Google App Engine. *Google code*. [Online] [Cited: Feb. 17, 2011.] <http://code.google.com/appengine/>.
- [5] About Pipes. *Yahoo! Pipes*. [Online] [Cited: Feb. 17, 2011.] <http://pipes.yahoo.com/pipes/>.
- [6] IBM Mashup Center. *IBM*. [Online] [Cited: Feb. 17, 2011.] <http://www-01.ibm.com/software/info/mashup-center/>.
- [7] *Persistent access control: a formal model for drm*. **Arnab, Alapan and Hutchison, Andrew.** New York : ACM, 2007. Proceedings of the 2007 ACM workshop on Digital Rights Management. pp. 41–53. 978-1-59593-884-8.
- [8] *Managing Digital Rights using Linear Logic*. **Barth, Adam and Mitchell, John C.** Washington : IEEE Computer Society, 2006. LICS '06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science. pp. 127 – 136. 0-7695-2631-4.
- [9] *LicenseScript: A Novel Digital Rights Language and its Semantics*. **Chong, Cheun Ngen, et al.** Los Alamitos : s.n., 2003. Third International Conference on WEB Delivering of Music (WEDELMUSIC'03). pp. 122 – 129. 0-7695-1935-0.
- [10] *A Formal Foundation for XrML*. **Halpern, Joseph Y. and Weissman, Vicky.** Pacific Grove : IEEE, 2004. 17th IEEE Computer Security Foundations Workshop (CSFW'04). pp. 251 – 265. 0-7695-2169-X.
- [11] *A formal foundation for XrML*. **Halpern, Joseph Y. and Weissman, Vicky.** 1, New York : ACM, February 2008, Journal of the ACM (JACM), Vol. 55, pp. 4:1 - 4:42. 0004-5411.
- [12] *A Logic for Reasoning about Digital Rights*. **Pucella, Riccardo and Weissman, Vicky.** Washington : IEEE Computer Society, 2002. CSFW '02: Proceedings of the 15th IEEE workshop on Computer Security Foundations. pp. 282 – 294. 0-7695-1689-0.
- [13] *Formal digital license language with OTS/CafeOBJ method*. **Xiang, Jianwen, Bjorner, Dines and Futatsugi, Kokichi.** Washington : IEEE Computer Society, 2008. AICCSA '08: Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications. pp. 652 - 660. 978-1-4244-1967-8.
- [14] *DRM interoperability analysis from the perspective of a layered framework*. **Heileman, Gregory L. and Jamkhedkar, Pramod A.** Alexandria : ACM, 2005. DRM '05: Proceedings of the 5th ACM workshop on Digital rights management. pp. 17 - 26. 1-59593-230-5.
- [15] *Abstract Interoperability between ODRL and MPEG-21 REL*. **Polo, Josep, Prados, Jose and Delgado, Jaime.** Vienna : s.n., 2004. Proceedings of the First International ODRL Workshop.

- [16] *Interoperability Challenges for DRM Systems*. **Schmidt, Andreas U., Tafreschi, Omid and Wolf, Ruben**. Ilmenau : s.n., 2004. International Workshop for Technology, Economy, Social and Legal Aspects of Virtual Goods. http://virtualgoods.tu-ilmenau.de/2004/Interoperability_Challenges_for_DRM_Systems.pdf.
- [17] *The Long March to Interoperable Digital Rights Management*. **Koenen, Rob H., et al.** 6, s.l. : IEEE, June 2004, Proceedings of the IEEE, Vol. 92, pp. 883 - 897. 0018-9219.
- [18] *Import/export in digital rights management*. **Safavi-Naini, Reihaneh, Sheppard, Nicholas Paul and Uehara, Takeyuki**. Washington DC : ACM, 2004. DRM '04: Proceedings of the 4th ACM workshop on Digital rights management. pp. 99 - 110. 1-58113-969-1.
- [19] *Enabler Release Definition for DRM; Draft Version 2.0 – 1 April 2004*. s.l. : Open Mobile Alliance, 2004. http://xml.coverpages.org/OMA-ERELED_DRM-V2_0_0-20040401-D.pdf.
- [20] *Open Digital Rights Language (ODRL) Version 2 Requirements*. 2005. <http://odrl.net/2.0/v2req.html>.
- [21] *MPEG-21 Rights Expression Language: Enabling Interoperable Digital Rights Management*. **Wang, Xin.** 4, Los Alamitos : IEEE Computer Society, October 2004, IEEE MultiMedia, Vol. 11, pp. 84 - 87. 1070-986X.
- [22] *Marlin Architecture Overview*. s.l. : Marlin Developer Community, 2006. <http://www.marlin-community.com/public/MarlinArchitectureOverview.pdf>.
- [23] *DRM as a Layered System, Proceedings of the Forth ACM Workshop on Digital Rights Management*. **Jamkhedkar, P. A. and Heileman, G. L.** Washington, DC : Association for Computing Machinery, Oct. 25, 2004.
- [24] **Jamkhedkar, Pramod A. and Heileman, Gregory L.** Rights Expression Languages. [ed.] Shiguo Lian and Yan Zhang. *Handbook of Research on Secure Multimedia Distribution*. Hershey : IGI Global, 2009, 1.
- [25] *The Problem with Rights Expression Languages; Proceedings of the Sixth ACM Workshop on Digital Rights Management (pp. 59-67)*. **Jamkhedkar, P. A., Heileman, G. L. and Martinez-Ortiz, I.** Alexandria, VA : Association for Computing Machinery, Oct. 30, 2006.
- [26] Main page. *Semantic MediaWiki*. [Online] [Cited: November 23, 2010.] <http://semantic-mediawiki.org>.